Junior Journal

Journal Name

OPEN ACCESS ARTICLE

LinkJVM 2.0 - Java on the KIPR Link

Regular Paper

Markus Klein[1,*] and Christoph Hackenberger[1]

[1]Vienna Institute of Technology (TGM)
* Corresponding author E-mail: m@mklein.co.at

Received D M 2014; Accepted D M 2014

DOI: 10.5772/chapter.doi

Abstract One major part of any robot is its software. The software brings the robot to life. So it is really important that the software is powerful but at the same time it should be maintainable as easy as possible, especially when the program is written by a team.

Java is an object oriented programming language, which is not that hard to learn. Unfortunately the KIPR Link does not support Java out of the box.

LinkJVM provides an open source full Java Runtime Environment(JRE) as well as an object oriented and easy to use library for controlling the robot.

The following paper shows a high and low level based implementation of a Java library for controlling the robot. This is archieved by wrapping the native libkovan. LinkJVM is also integrated into the KIPR Link's build system which allows easy and user-friendly deployment. Further this papers describes how to write programs in Java with the LinkJVM and how to get them running on the KIPR Link.

Keywords Java, JVM, Botball, Library, Framework, Wrapper, KIPR Link, Libkovan, Robotic, JNI, SWIG, JamVM, GNU Classpath, Open Source

1. Introduction

The purpose of Botball[1] is to motivate students for building and programming autonomous robots. Most new students to Botball do not have any experience with programming. Therefore the way of developing the robot software has to be very beginner friendly and easy to use, but especially in Austia where most botball students come from technical schools there is also a need for high level languages.

So what is the best language for Botball? The authors do not think there is a best choice. Every language has its own advantages and disadvantages. C for example is a very powerful fast and processor-near language so it is quite good for a robot controller doing a lot of basic things such as controlling servos and motors, reading sensor values and so on. This is good for students without programming experience which only implement simple algorithms, but for more experienced students with more complex code C is not the right choice. While libkovan mostly avoids the real challenges in coding C, when implementing complex algorithms there is no way around pointers and memory management.

Another well-known possibility is Java[2]. It is much easier to learn and also pretty powerful and reliable. Of course it is not as fast as compiled languages such as C or C++, but in the most cases it will make no real difference. Java is a higher level language than C which makes it in general easier to implement complex algorithms. It provides automatic garbage collection to handle memory management and offers an excellent set of basic data structures such as lists, queues and maps as well as a huge standard library including easy to use APIs for network programming, multithreading and much more. In C everything has to be built from scratch. The object-oriented paradigm is also very helpful to structure the program to improve the maintainability, especially in bigger projects. Moreover Java is compiled into cross

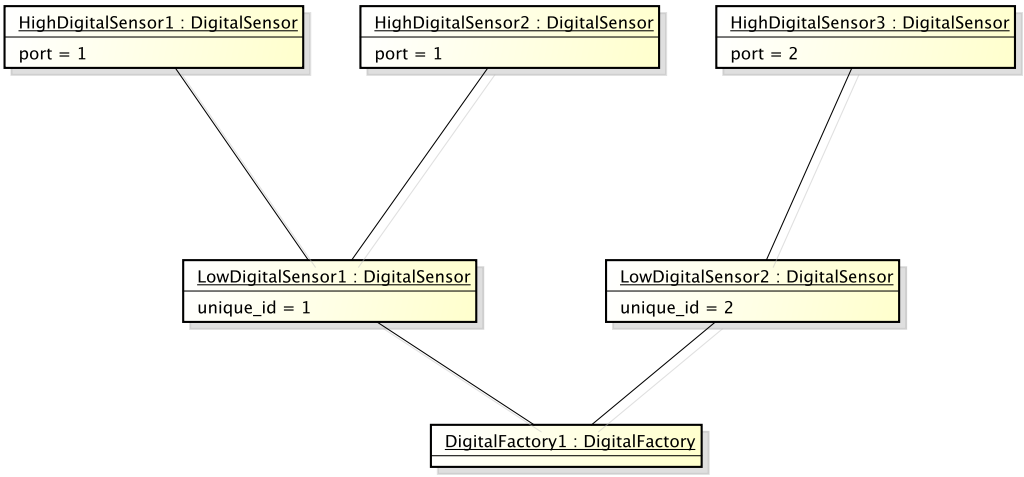

Figure 1. Object-Diagram showing a typical structure of a LinkJVM application

platform bytecode instead of target specific machine code, so Java programs do not need to be compile directly on the Kiss Institute of Practical Robotics[3] (KIPR) Link[4].

In most schools in Austria students get started with programming using Java and not C or C++. Therefore the authors would consider themselves as a good Java programmer, but they never wrote a bigger project using C nor C++.

Further alternatives are event-based frameworks using a scripting language such as node.js[5] or in general scripting languages, but these can be implemented very easily, because the Java Virtual Machine JVM[6](JVM) supports a huge amount of additional languages including javascript, scala and clojure.

2. State of the Art

LinkJVM 1.0[7] was published at the Global Conference on Educational Robotics 2013[8] (GCER 2013). While the Java environment JamVM[9] and GNU Classpath[10] works pretty well the robot library is not that good. At that time LinkJVM 1.0 wraps over libkovan's[11] C bindings which makes the libary structure pretty poor, because LinkJVM 1.0 first calls the native C function and then the native C function manages the C++ objects. Though this approach is easier to implement, but it also comes with some downsides. The most important of them is that there is no possibility to write a thread safe library using this approach, because libkovan itself is not thread-safe. Moreover LinkJVM 1.0 offers only a very small API with much less features that libkovan does and LinkJVM's 1.0 vision system contains also just one camera class. Therefor the authors decided to write a new library from scratch, which will fix all this issues.

3. Design Approach

The framework of the LinkJVM consists of one high and one low level part. The low level part of the LinkJVM wraps the existing C++ library of libkovan, the high level now wraps the low level part and represents the API for the user. So the user only uses the high level part of the LinkJVM and should not come in contact with any low level things.

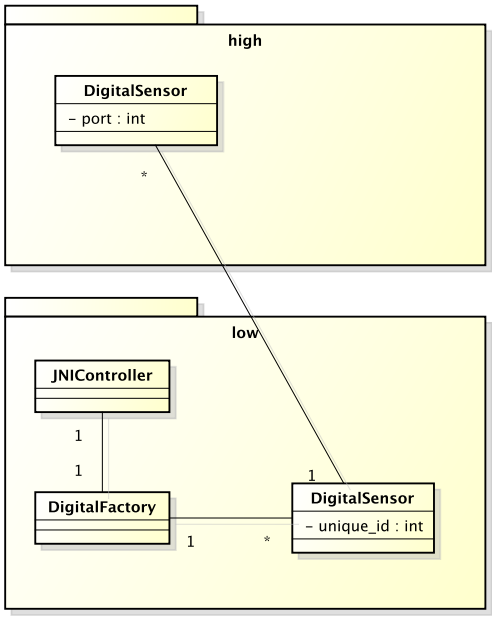

Figure 2. Class-Diagram showing the releationships or the different high and low level components

For example the user has a digital sensor like a lever on his robot and now wants to use this sensor in his program. He will now create a new high level DigitalSensor object, which only uses one low level DigitalSensor object in the background (see Figure 2) that again uses the C++ implementation of a DigitalSensor.

The user can create as many high level objects of the same sensor, motor, servo,... , which will all uses the same low level object with the specified unique identifier, which is for example the port of a sensor. If the user creates a new high level object with a unique identifier that is unequal to an existing low level object the appropriate factory would create a new low level object with the specified unique identifier. (see figure 1)

Both the low and the high level parts are described in detail in the following subsections:

3.1. Low Level

The low level part of the framework contains the Java side libkovan wrapper. Every object of a low level class contains also a native C++ object. This classes must be instantiated using the corresponding multiton factory, which means that every low object has a unique identifier as described in section Design Approach. The low level part also provides the JNIController class which holds a singleton of every factory as well as one static singleton object of itself. Most low level parts are automatically generated wrapper classes using the Simplified Wrapper and Interface Generator(SWIG[12]). This software generates the Java and the C++ wrapper out of an interface file with all class definition that should be wrapped.

3.2. High Level

The high level part contains, as already mentioned before, the object oriented API for the user. Internally every time the user creates a new high level object, the high level object requests a new low object. (See figure 2 and figure 1).

4. LinkJVM - API

The API of LinkJVM (or the robot library) provides a object oriented, well documented and easy to use and maintainable library for controlling the KIPR Link and all its connected devices. The API contains classes for the following components:

- iRobot Create[13]
- motors and servos
- analog and digital sensors
- hardware and software buttons
- vision system
- depth camera (experimental)
- AR.Drone 1.0/2.0[14]

The most important components of the framework are described in detail in the following subsections.

4.1. Botball Class

The Botball Class contains some basically static methods like shutDownIn, waitForLight or msleep. The next listing shows the usage of these methods:

Listing 1. Usage of the Botball class

```
//Terminates the program in 120 seconds
Botball.shutDownIn(120);
//Wait for the start light on sensor port 0
Botball.waitForLight(0);
//Sleep for 100 ms
Botball.msleep(100);
```

4.2. Motors/Servos and Sensors

LinkJVM's robot libary provides classes for the different types or motors, servos and sensors. The following listing shows the usage of motors and sensors:

Listing 2. Basic usage of sensors and motors

```
public static void main(String[] args){
  Motor leftMotor = new Motor(0);
  Motor rightMotor = new Motor(1);
  DigitalSensor bumpSensor =
  new DigitalSensor(0);
  leftMotor.turn(100);
  rightMotor.turn(100);
  while(!bumpSensor.getValue()){
    Botball.msleep(100);
  }
  leftMotor.freeze();
  rightMotor.freeze();
}
```

The example above shows a very simple program, which lets the robot drive forward until a digital sensor which is used as bump sensor bumps.

4.3. Create

The iRobot Createis a very good example to show the advantage of the usage of an object oriented programming language like Java. The following example shows the usage of the iRobot Create:

Listing 3. Usage of the iRobot Create with LinkJVM

```
public static void main(String[] args) {
  Create create = new Create();
  create.connect();
  SideButton sb = new SideButton();
  BButton bb = new BButton();
  bb.setText("Start");
  while(!bb.getValue());
  while(!sb.getValue()) {
    if(create.getLeftBump())
      create.spinClockwise(250);
    else if(create.getRightBump())
      create.spinCounterClockwise(250);
    else
      create.driveStraigth(400);
    Botball.msleep(50);
  }
  create.stop();
  create.disconnect();
}
```

With this code it is possible to do a room tour with the iRobot Create. After the start button was pressed the Create starts moving until it bumps, then it turns and moves on in another direction. The program runs until the side button on the KIPR Link is pressed.

4.4. Vision System

The LinkJVM's vision system consists basically of 3 major classes(see figure 3):

- **ImageProcessor**: The ImageProcessor processes the image to color or QR blobs with a given CameraConfig and a given channel number and proviedes afterwards the CameraObjects.
- **CameraObject**: One CameraObject represents an object tracked by an image processor. It contains information like the bounding box or the center point about the blob.
- **CameraConfig**: An CameraConfig object contains several configuration attributes for an ImageProcessor.

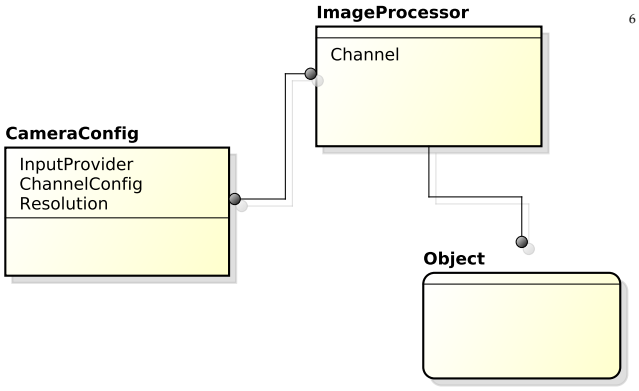

Figure 3. Overview over LinkJVM's Vision System

Listing 4. Basic usage of the vision system

```
public static void main(String[] args) {
  Create create = new Create();
  create.connect();
  ImageProcessor camera =
  new ImageProcessor(new CameraConfig
  (Resolution.MED_RES), 0);
  camera.openCamera();
  SideButton sb = new SideButton();
  while(!sb.getValue()) {
    camera.update();
    if(camera.getObjectCount() > 0) {
      Rectangle rec =
      camera.getBoundingBox(0);
      if(rec.getWidth() > 80){
        if(rec.getCenter().getX() < 290){
          create.driveDirect(100, 0);
        }else
        if(rec.getCenter().getX() > 350){
          create.driveDirect(100, 0);
        }
        Botball.msleep(500);
      }
    }
    create.stop();
  }
  create.disconnect();
  camera.close();
}
```

4.5. Debugger

The LinkJVM-Debugger allows the user to send debug message over the network to a debugger client. Therefore the API provides a debugger class which connects establishes a connection to a client.

Listing 5. Example usage of the LinkJVM-Debugger

```
public static void main(String[] args){
  Debugger debugger = new Debugger(IP);
  debugger.write("Some text!");
  debugger.writeln("Some line!");
  debugger.close();
}
```

5. Java Environment

LinkJVM includes a Java Runtime Environment[15] (JRE) which consists of a lightweight Java Virtual Machine(JVM) and GNU Classpath an open source implementation of the Java core classes. So LinkJVM 2.0 uses basically the same Java environment than LinkJVM 1.0 does, but in addition LinkJVM 2.0 offers the »Eclipse Java compiler«as Java compiler (javac) and the »jar«for packaging multiple class files. Using these tools Java programs can be directly compiled and packaged on the KIPR Link, but since class and jar files contain cross platform bytecode, it is not necessary to compile the programs on the KIPR Link. (See section Developing Tools)

6. Integration into KIPR Link's build system

The KIPR Link is based on a build framework called OpenEmbedded [16] and its accociated build system »bitbake«which provides a cross compile environment for embedded Linux[17]. Unfortunately the KIPR Link does not work with the latest version of these tools, so building the firmware or adding packages is very hard. Furthermore OpenEmbedded only works on Linux machines and also requires knowledge of basic and moderately complex terminal commands. Fortunately with the excellent help of KIPR staff, the authors managed to set up a build server and compile a LinkJVM package.

6.1. LinkJVM's build system

As already mentioned before LinkJVM's source consists of a C++ and a Java part. To combine both build processes using only one build system the authors decided to use CMake[18] for compilation. CMake is an open source, cross platform build system which supports C++ as well as Java and automatically generates makefiles.

For integration into KIPR Link's build system the authors added an meta-layer which provides a bitbake recipes for building LinkJVM.

6.1.1. Deployment

Using bitbake LinkJVM is compiled and packaged into a ipk software package which can be easily installed on the Link directly via BotUI.

It is also possible to directly add LinkJVM to a firmware image and in the future LinkJVM may also be integrated into the offical Link firmware.

7. Developing Tools

Because the code the user writes with the LinkJVM library uses the programming language Java, it is not possible to use the KISS IDE from KIPR because it only supports C or C++ code.

The authors decided to leave the choice of how to write the code at the user. It is possible to write the code with an IDE like Eclipse or Netbeans or just write it in a simple text editor and compile the source code over the console.

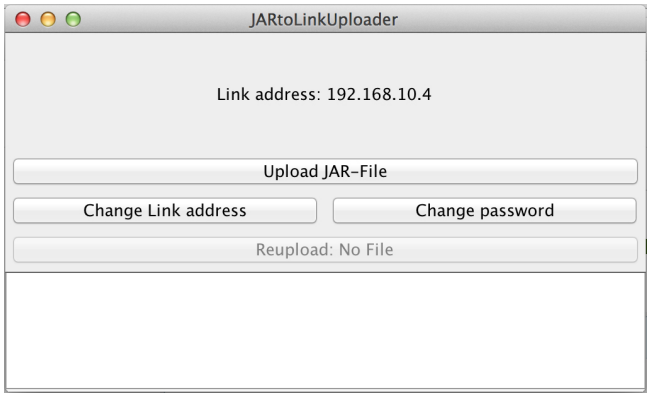

Figure 4. LinkJVM-Uploader

At the beginning it was only possible to upload/run the user's program via scp/ssh to/on the KIPR[3] Link[4]. Running a program over ssh is not tournament-conformal, so the only thing that could be done was to write a C wrapper, which executes your Java program.

So the next logical thing the authors have to think about is how to get the compiled program on the KIPR Link and make it run able there in a for the user easy way? They first thought about a plugin for the Eclipse IDE but this would extremely limit the user of how to write the code, which like already mentioned the authors wanted to leave the choice at the user. The authors decided to write an independent program, which is called LinkJVM-Uploader or JARtoLinkUploader (see Figure 4), to upload .jar files to the Link. This program also automatically creates C wrappers, which executes the .jar files and which are shown in the menu point »Programs«on the KIPR Link. All this is done over ssh and scp connections in the background.

8. Conclusion

LinkJVM 2.0 comes with a completely new robot library which fixes many problems of LinkJVM 1.0[7]. Now the user does not have care of the thread safety and the vision system provides an excellent API for using the KIPR[3] Link's[4] camera. Since LinkJVM has also been integrated into the KIPR Link's build it is incredibly easy to install and upgrade which makes it pretty attractive for Botball students. Although the design approach shown in this paper works very good, it is still not perfect, especially because developing on the core of LinkJVM requires advanced knowledge of the Java Native Interface(JNI), which makes it pretty hard for students to fix bugs on their own. Moreover since there are very few people with a build server compiling LinkJVM is nearly impossible.

So the next logical step would be to make LinkJVM's robot library a pure Java library and replace the low level part with real implementation. This could be archieved pretty easy, because the most parts of libkovan does nothing else than communicates with the kovan kmod via UDP. The kovan kmod is a kernel module which provides an interface to KIPR Link's Field Programable Gate Array (FPGA). Using this approach, LinkJVM could also be built without an build server, because it would only consist cross platform bytecode.

9. References

[1] Botball, Standards-Based Educational Robotics Program, http://www.botball.org, 2014

[2] Java, Oracle, http://www.java.com/en/, March 2014

[3] KISS Institute for Practical Robotics, http://www.kipr.org, 2014

[4] Link, Robot Controler, http://www.kipr.org/products/link, March 2014

[5] Node Javascript, http://nodejs.org/, March 2014

[6] JVM, Java Virtual Machine, http://en.wikipedia.org/wiki/Java_virtual_machine/, March 2014

[7] Klein Markus: »Java on the KIPR Link«, http://files.kipr.org/gcer/2013/proceedings/Klein_Java_KIPR_Link.pdf, 2013

[8] GCER, Global Conference on Educational Robotics, http://www.kipr.org/gcer, 2014

[9] JamVM, http://jamvm.sourceforge.net/, 2010.

[10] GNU Classpath, open source Java core class implementation, http://www.gnu.org/software/classpath/, March 2014

[11] McDorman B.: libkovan C standard library, https://github.com/kipr/libkovan, March 2014

[12] SWIG, Simplified Wrapper and Interface Generator, http://www.swig.org, March 2014

[13] iRobot Create, Programable Robot, http://www.irobot.com/us/learn/Educators/Create.aspx, March 2014

[14] AR.Drone, Parrot, http://ardrone2.parrot.com, March 2014

[15] JRE, Java Runtime Environment, http://www.oracle.com/technetwork/java/javase/downloads/, March 2014

[16] OpenEmbedded, build framework for embedded Linux, http://www.openembedded.org/wiki/Main_Page, March 2014

[17] Braden McDorman, Joshua Southerland: »A Look Inside the KIPR Link«, http://files.kipr.org/gcer/2013/proceedings/McDorman_A_Look_Inside_the_KIPR_Link.pdf, 2013

[18] Cross Make, cross-platform, open-source build system, http://cmake.org/